IBM Parallel Port FAQ/Tutorial

Version 0.96 9/1/94

by Zhahai Stewart

(Note, the links to the author and for this page were dead so I have saved a copy to my site for posterity)

Interfacing the IBM PC Parallel Printer Port

1. Overview

This document is called an FAQ because it answers many commonly asked questions about the IBM parallel port, but it is formatted more as a brief tutorial. Read it twice before asking for more info; some stuff comes late. Most of this I puzzled out for myself from various documention and experiences. The IBM documentation has some errors; I've had to cross correlate various sources including schematics to get a consistent and workable picture. Since the first version, others have contributed information, proofreading, and suggestions (see acknowledgements). Starting with the original IBM PC, IBM defined a standard parallel printer port which has become very widespread. This port uses a female DB-25S connector on the computer, and a special male DB-25P to Centronics male 36 pin "IBM Printer cable" is used to connect to standard Centronics parallel printers. This DB-25 connector is possible because about half of the Centronics pins carry just electrical ground. The original definition was embodied in the "IBM Printer Adaptor", and the "IBM Monochrome Display and Printer Adaptor" cards.

2. Conventions used in this document.

An electrical "high" (on a pin or line) is TTL high, +2.4 to +5 volts. An electrical "low" is TTL low, 0 to +0.8 volts. A data high is a 1, a data low is a 0. The connection between data (eg: a register bit) and pins is direct if a data 1 is associated with an electical TTL high, and inverted if data 1 is associated with TTL low. An overall connection (data to TTL to data) is considered direct if outputing a 1 produces a 1 on input at the other end, or inverted if outputting a 1 produces a 0 on the other end. Pin labels, like "-Strobe" or "+Busy", are as defined by the printers and by IBM. The prefix "-" (or draw a line over the name) implies that the signal is "active low"; that is, that when the signal is in its active state when electically low. The "+" prefix means "active high", just the opposite. I have labeled the Data Out register bits as D0 to D7, with D0 being the least significant bit and D7 the most signficant. The Control Out bits are labeled C0 to C3 (for the ones which go to pins) and C4 (for IRQ enable), and maybe C5 (for bidirection ports only, controls direction). The Status In bits are labeled S3 to S7 (corresponding to data and CPU bit positions). Often I will suffix C0 to C3 and S3 to S7 with a "+" or "-" as a reminder of whether or not that register bit is inverted as compared to the output or input pin it is associated with; "-" is for inverted, of course. All the Data Out bits are direct (not inverted); likewise data in for bidirectional printer ports, but I have not bothered to suffix a "+". Hexadecimal numbers are prededed by "0x", the C convention. A "tristated" or disabled electical output basically disconnects the output from the line or pin (high impedance), neither driving it high nor low.

3. Addresses, naming, BIOS and DOS

IBM defined three standard port base addresses (in 80x86 IO address space). The Printer Adaptor could

use base address 0x378, or later 0x278, while the Monochrome Display and Printer Adaptor used base address 0x3BC. The IBM BIOS defines RAM space for 4 parallel printer port base addresses, stored as 4 16 bit words starting at main memory address 0x408. During bootup, the BIOS will check for parallel printer ports at base addresses 0x3BC, 0x378, and 0x278, in order, and store the base addresses of any that are found in consecutive locations in this table. Unused entries may be 0, or some BIOSes fill them with the first port address found. Some software may ignore this table, but it is used by at least the BIOS itself (eg: INT 17, Printer I/O), and by DOS, as described below. The BIOS detects these ports by writing 0xAA to the Data Out register (at I/O address Base+0), reading the Data Feedback register (same address), and deciding there is a port installed if it reads 0xAA. This could be confused if any lines are externally pulled up or down (or if the port defaulted to tristate, or if there is another readback device register at that address). The BIOS also counts the number of parallel ports it found and stores this count in the upper two bits of the byte at 0x411 (yes, the table can hold up to 4 entries, but the BIOS equipment flag printer count only goes to 3). Warning: Just before this table there are 4 words at 0x400 which contain up to 4 entries for base addresses for serial COM ports. At least some serial port software is known to store more than 4 entries, thus overlapping the parallel port table. Hopefully this is rare! DOS (MSDOS and IBM DOS) maps these as LPTn devices. Unlike COMn devices and comm ports, the name mapping varies depending on whether or not there is a Monochrome Display and Printer Adaptor card or not. The first entry in the BIOS table at 0x408 becomes LPT1, the second entry LPT2, and the third entry LPT3 (if there are that many). The DOS device "PRN" is really software alias for another port, by default LPT1; use the MODE command to change this aliasing. The following table has "typical" assignments. Note that by swapping the entries in the BIOS table at 0x408, you can change which physical ports are assigned to LPT1, LPT2, etc. Several "printer swap" programs do just that.
Typical Assignments

```
Addr    MDPA    no MDPA

 0x3BC  LPT1    n/a     Monochrome Display and Printer Adapter (MDPA)

 0x378  LPT2    LPT1    Primary Printer Adapter

 0x278  LPT3    LPT2    Secondary Printer Adapter


Name    MDPA    no MDPA

 LPT1   0x3BC    0x378

 LPT2   0x378    0x278

 LPT3   0x278     n/a
```

4. Registers addresses within the parallel printer port:

```
Port          R/W IOAddr   Bits   Function

----------        ------   -----  ----------------

Data Out      W   Base+0   D0-D7  8 LS TTL outputs

Status In     R   Base+1   S3-S7  5 LS TTL inputs

Control Out   W   Base+2   C0-C3  4 TTL Open Collector outputs
```

"	"	"	C4	internal, IRQ enable
"	"	"	C5	internal, Tristate data [PS/2]
Data Feedback	R	Base+0	D0-D7	matches Data Out
Control Feedbk	R	Base+2	C0-C3	matches Control Out
"	"	"	C4	internal, IRQ enable readback

The Feedback registers are for diagnostic purposes (except in bidirectional ports, where Data Feedback is used for data input; and the IRQ enable C4).

5. Pin signals and register bits

<= in	DB25	Cent	Name of	Reg	
=> out	pin	pin	Signal	Bit	Function Notes
------	----	----	-----------	-----	-------------------------
=>	1	1	-Strobe	C0-	Set Low pulse >0.5 us to send
=>	2	2	Data 0	D0	Set to least significant data
=>	3	3	Data 1	D1	...
=>	4	4	Data 2	D2	...
=>	5	5	Data 3	D3	...
=>	6	6	Data 4	D4	...
=>	7	7	Data 5	D5	...
=>	8	8	Data 6	D6	...
=>	9	9	Data 7	D7	Set to most significant data
<=	10	10	-Ack	S6+ IRQ	Low Pulse ~ 5 uS, after accept
<=	11	11	+Busy	S7-	High for Busy/Offline/Error
<=	12	12	+PaperEnd	S5+	High for out of paper
<=	13	13	+SelectIn	S4+	High for printer selected
=>	14	14	-AutoFd	C1-	Set Low to autofeed one line
<=	15	32	-Error	S3+	Low for Error/Offline/PaperEnd
=>	16	31	-Init	C2+	Set Low pulse > 50uS to init
=>	17	36	-Select	C3-	Set Low to select printer

```
==      18-25   19-30,  Ground

                33,17,16
```

Note: Some cables, ports, or connectors may not connect all grounds. Centronics pins 19-30 and 33 are "twisted pair return" grounds, while 17 is "chassis ground" and 16 is "logic ground". "<= In" and "=> Out" are defined from the viewpoint of the PC, not the printer. The IRQ line (-Ack/S6+) is positive edge triggered, but only enabled if C4 is 1. Here's the same data grouped for ease of reference by Control Out and Status In registers and pins. (Data Out is straightforward in previous table).

```
<= in   DB25   Cent   Name of     Reg

=> out  pin    pin    Signal      Bit     Function Notes

------  ----   ----   --------    ---     -----------------------------

=>       17     36    -Select     C3-     Set Low to select printer

=>       16     31    -Init       C2+     Set Low pulse > 50uS to init

=>       14     14    -AutoFd     C1-     Set Low to autofeed one line

=>        1      1    -Strobe     C0-     Set Low pulse > 0.5 us to send


<=       11     11    +Busy       S7-     High for Busy/Offline/Error

<=       10     10    -Ack        S6+ IRQ Low Pulse ~ 5 uS, after accept

<=       12     12    +PaperEnd   S5+     High for out of paper

<=       13     13    +SelectIn   S4+     High for printer selected

<=       15     32    -Error      S3+     Low for Error/Offline/PaperEnd
```

6. Electrical

See also the tutorial section below on TTL outputs. The Data Out pins were orginally driven by a 74LS374 octal latch, which could source 2.6 mA and sink 24 mA. There were 0.0022uF capacitors between each line and ground to reduce transients. The manual warns "It is essential that the external device not try to pull these lines to ground", as this might cause the 74LS374 to source more current than it could handle without damage. The Feedback input port for the Data Out register consisted of a 74LS244 tri-state buffer; it is uninverted. Note that this port is only for diagnostics - if at any time the feedback buffer for the data port does not match the data being output, there is an error (eg: a line stuck high or low). Exception: bidirectional printer ports allow the 74LS374 (or equivalent) driver chip to be tri-stated, which makes the data feedback port into a legitimate unlatched input port. The Control Out pins were driven by 7405 inverting open collector buffers, pulled to +5 volts via 4.7K resistors. All data lines except C2 are inverted before going to output pins; data line C2 is double inverted before going to pin 16 (ie: is not inverted). The Feedback input for the Control Out register also inverted all but C2, which was passed through uninverted. It is possible to use some or all of the control out port bits for input, by programming the corresponding control out to high (remembering the inversion on C0, C1, and C3), in which case the open collector outputs are pulled high by the 4.7K resistors; any externally applied high will retain the high state, while an externally applied low will pull down the electical level to low. This can be read via the corresponding feedback bit(s). If either the output from the control register, or an externally applied signal, are low, then the input will be low. Remember the inversions between this electrical level and the bits, tho. The Status In register is inverted for only pin 10, register bit S7. ESD (from Steven M. Scharf) LSI implementations of the parallel port have often had serious ESD (electrostatic discharge; includes static electricity) problems. National and other high-end manufacturers have added extensive anti-ESD circuitry to the parallel port signal lines; on cheap parallel port designs on some other SuperIO clones ESD can easily destroy the parallel port circuit when you turn the printer on when the computer is off or the printer is on a switchbox.

7. Timing

The original IBM Printer Adapter which came out along with the PC (floppy disks, pre XT) was built from TTL MSI parts. They (and their clones) were fast enough to work with the later AT bus, now called ISA, and still do. I've yet to hear of a TTL printer adapter which is too slow for an ISA bus. Partly this is because it was simple and fast in bus access (unlike the serial port 8250 chip), and partly this is because even fast 486's slow their ISA bus down to 8-11 MHz typically.

8. How to Print to a standard printer.

This is as defined for early IBM printers, and more or less compatible with most others that use the Centronics 36 pin interface.

```
Wait for not +Busy (+Busy low)

Set Data Out bits

    at least 0.5 uS delay

Pulse -Strobe low for at least 0.5 uS

    hold Data Out for at least 0.5 uS after end of -Strobe pulse

Some time later, printer will pulse -Ack low for at least 5 uS

Printer may lower +Busy when it raises -Ack at end of pulse

Set other Control outputs or check Status inputs as desired.
```

9. BIOS Printer Support

The PC ROM BIOS detects and initializes the printer ports on bootup, and provides printer support via INT 17 (and indirectly uses this in INT 5 "Print Screen"). The detection and building of a printer port base address table is described in the Soft Addresses section above. To control the printer via the BIOS, call INT 17 with the printer number in DX (0 to 3, indexing into the four entry table at 0x408 mentioned in the Soft Addresses section above), and the desired function in AH. Functions:

```
AH = 0: print the character in AL

AH = 1: reinitialize the port, return status in AH

AH = 2: return status in AH
```

BIOS operation notes follow.

```
The BIOS normally keeps the Control register value as 0x0C:

        C5 = 0  =>  output enabled (for bidirectional only)

        C4 = 0  =>  IRQ disabled

        C3 = 1  =>  -Select pin low     (Printer Selected)

        C2 = 1  =>  -Init pin high      (Printer not being initialized)

        C1 = 0  =>  -AutoFeed pin high (No Auto Feed)

        C0 = 0  =>  -Strobe pin high    (No Stobe)
```

The BIOS prints a character (function AH = 0) as follows:

- (Uses only Status in S7/+Busy and Control out C0/-Strobe)
- Write character to Data Out (D0-D7)
- Wait for bit S7 to go to 1 (-Busy pin to go high/low?)
- Timeout and return if this takes too long
- Set C0 to 1 for a few microseconds (-Strobe pin pulsed low)

This assumes that the printer was already ready. The BIOS handles reinitializing the printer (function AH = 1) thus:

- Set C2 = 0 for a few hundred microseconds (-Init pin pulsed low)

The BIOS returns status in AH (functions AH = 1 or 2) is as follows:

```
 ___________________________________

 | 7 | 6 | 5 | 4 | 3 | 2   | 1 | 0 |

   |   |   |   |   |  not used  |___ 1 = "Time Out" (software)

   |   |   |   |   |________________ 1 = "I/O Error" (inv C3+; -Error pin low)

   |   |   |   |____________________ 1 = "Selected"  (C4+; +SelectIn pin high)

   |   |   |________________________ 1 = "Paper Out" (C5+; +PaperEnd pin high)

   |   |____________________________ 1 = "Acknowledge" (inv C6+; -Ack pin low)

   |________________________________ 1 = "Not Busy"  (C7-; +Busy pin low)
```

The high 5 bits of this are essentially the Status In register from the printer port, with bits 6 and 3 inverted (ie: XORed with 0x48). The low bit is generated by BIOS software after a timeout. In the original PC, timeout delays were generated by delay loops.

10. IRQ's

The primary IBM Printer Adapter (base 0x378) and the IBM Monochrome Display and Printer Adapter (base 0x3BC) are allocated hardware interrupt 7 (IRQ 7), which is wired to INT 0x0F. If bit C4 of the Control Out register is high, then input line "-ACK" (DB 25 pin 10, Status Register bit S6) will be wired to the IRQ 7 bus line. A rising edge (low to high) on -Acknowledge will be cause a rising edge on IRQ 7, which will trigger an interrupt if that IRQ is enabled in the 8259 interrupt controller chip. (Note: the MCA bus is level sensitive rather than edge triggered). Both cards use the same interrupt, and the ISA bus does not share interrupts correctly, so no more than one of these ports should be enabled at the same time (or the 74LS125 chips driving the PC bus will fight). Secondary Printer Adapters (base 0x278) are supposed to use IRQ 5 (INT 0x0D) for their interrupt; note that on the PC (pre-AT) this would have conflicted with the floppy disk controller. This IRQ is even more commonly "stolen" for other usage: EGA, soundcards, network cards, etc. The idea was apparently to use this IRQ as part of the printer driver, much as the serial port IRQs can be used as part of comm drivers. Unfortunately, typical timing on printers did not allow this, so most (nearly all) DOS software does not use the printer port IRQ for driving printers. It's often considered "up for grabs" and some soundcards like to use IRQ 7. Some other operating systems may use it for the printer port, as may some parallel port transfer programs.

11. Bidirectional Ports

The IBM PS/2 series added one feature to the parallel ports: bidirectional support. This was done by using one more bit in the Control Out register to control tristating of the Data Out port. When bit C5 is 0, the Data Out port operates as it did for earlier parallel ports (driver enabled); when C5 is set to 1, though, the Data Out port is tristated, which means that it is essentially electically disconnected from the pins (high impedance state), not driving them high nor low. This allows the data feedback register to correctly read any externally applied TTL signals on the corresponding data pins, effectively making this an input port. Interestingly, the original IBM Printer Adapter, and many early clones, had everything needed to become bidirectional. The 74LS374 chip has a chip enable pin which can tristate the 8 data outputs, but it is usually wired to ground to always enable the output drivers. The 74LS174 latch used for C0-C4 is actually a hex latch and processor data bus line 5 is attached to the sixth latch input; that bit IS latched when you write to the Control Out register - but the output (Q6 on the 74LS174) is not used by anything. If you cut the trace from pin 1 (/OE) of the 74LS374 to ground, and connect this pin instead to pin 15 (Q6) of the 74LS174, voila - you have a PS/2 compatible bidirectional parallel port. This only works on the older discrete TTL parallel ports, including the IBM Printer Adapter and at least some clones of that era. One theory is that IBM had a bidirectional port in mind initially, but decided to omit that aspect at the last moment (even while still latching the C5 bit from the processor into the LS174 hex latch). There are now PS/2 compatible third party bidirectional parallel ports. The cost for a TTL bidirectional port should have been identical, but now that parallel ports are done with one LSI chip (or a fraction of one), this will require an appropriate LSI chip. Apparently some of them now have bidirectional ports, some do not. Note that some people have for many years been using the standard (not bidirectional) data out port as an input port. They output data 0xFF, that is, all high. Any pins which are externally pulled low via TTL or switches will probably read as low (0) in the data feedback register, because TTL low (sinking current) tends to be stronger than TTL high (sourcing current), and tend to "win" when there two drivers fighting on the same line. An external high signal, or no signal, will allow the pin (and data bit) to remain high (1). This is NOT reccommended, as it stresses the 74LS374 (or equivalent) beyond its safe margins and could cause chip failure. However, some people report doing this for years without problem. Note that all recent parallel ports are implemented with single chip LSI controllers rather than the MSI TTL originals, which means that they may have different drive capacities, electical margins, and replacement costs, should you attempt this approach. Sometimes the LSI controller also includes other functions, such as serial ports or a monochrome display port; if it overheats, more than just the parallel port could be damaged. Beware. However, if you have an old TTL parallel port (with a 74LS374 chip, preferably socketed), it may be easy and cheap to replace if that chip should die. I bought TTL parallel port card clones, fully socketed (all chips!), for about $15 mail order about 5 years ago. The bidirectional conversion was a trace cut and a wire jumper. The 74LS374 was easily replaced, if it were damaged or if one wanted a better chip like the 74ACT374. You could even strap it for any base address (including non standard ones, so as not to conflict). I wish I could get more; there must have been scads of these on the market. You may find old TTL printer ports used; latch onto them if you want to experiment with parallel port interfaces (pun acknowledged). One last electrical note: if you use any sort of bidirectional parallel port for inputs, be sure to tristate it whenever it is connected to external devices which will drive it. When I'm running it this way, I put a small routine in my autoexec.bat which tristates the port in question on every bootup. Or use inline resistors to at least limit the current to a safe value. Steven M. Scharf notes that some higher end parallel port chips "lock" the direction control bit C5 to keep certain naive software from accidentally changing the port direction. In particular, he says that "PTR bit 7" of the National Semiconductor SuperI/O chip must be 1 before you can change the direction control bit. I don't have documentation on this chip, so I'm not sure what register PTR is; for now, consider this just a warning. He says that other chips may lock the direction control differently.

12. Enhanced Ports

There are at least two specs for further enhanced parallel ports, the EPP (Enhanced Parallel Port) and the ECP (Enhanced Capability Port), with the latter being more ambitious. I would appreciate any specs that can be forwarded to me (or references) on either of these. Steven M. Scharf gave this table:

```
Different Parallel Ports on NSC Parts

----------------------------------------

PC87311 Bidirectional Parallel Port (16450 UARTS)

PC87312 Bidirectional Parallel Port (16550 UARTS)

PC87322 Enhanced Parallel Port (EPP) (16550 UARTS)*

PC87332 Enhanced Capabilities Port (ECP) also EPP 1.7 and 1.9 (16550 UARTS)*


*Floppy controller signals can be redirected out the parallel port pins

via an internal multiplexor. This feature is used in some sub-notebooks

with external floppy drives
```

13. Copy Protection Dongles

(From info provided by Steven M. Scharf) Copy protection dongles typically watch the data lines for patterns of data without any -strobe signal (pin 1, controlled by C0-). That is, the copy protection software writes various patterns to the data lines without every pulsing the -strobe line. Without a pulse on -strobe, any printer also attached to the port should ignore this data. Not all dongles work on all ports; this strobeless method is a bit more temperamental than regular printer output. Steven M. Scharf: "I heard one dongle manufacturer complain to me about a parallel port from Taiwan that tristated all the data lines between every write to the data lines. This presented a pattern of FF to the dongle, but had no effect on the printer since there was no strobe. In this case the software was writing a sequence of bytes to the dongle and it didn't work due to the FF in between each expected real byte." "The dongle also draws power from the signal lines--a definite no-no. Dongles should be designed to operate all the way down to the minimum TTL low or at least to minimum Vout High of 2.4V. If your dongle doesn't work but your printer works fine then it is almost certainly the fault of the dongle--not the parallel port. Software with an incompatible dongle to the parallel port on a machine will not be usable on that machine--one more reason to not penalize the legitimate buyers of software."

14. Other devices:

From: geo@netcom.com (George Pontis) "Another area that might be of interest in your document would be some comment on the parallel port extenders. I have a xmit/rcv pair from LinkSys that I bought from Fry's Electronics for about $70. They convert the parallel signal to a serial data stream, using the signal and control lines for power. My set was working fine until I added a hardware dongle for an expensive Windows application. Then, printing ceased to work reliably. I took the transmitter apart and partially traced the schematic. They have used 7 diodes to suck power from pins 13, 14, 15, 17, 1, 2, and 3. Also, they connected pins 15 (ERR) to 16 (INIT). The strobe line is coupled in to a flip-flop, which starts clocking the parallel loaded data." Comment from Zhahai - with both a dongle and a parallel port extender trying to draw power from the port's data or control (not power) lines, it's not surprising if things don't always work! It must be really pushing the specs.

15. Transferring Data Via Ports

There are three basic ways to wire IBM printer ports together to transfer data between them. The most common is to wire D0 - D4 (or D3-D7) from one port to S3 - S7 from the other port, and vice versa. In this way, 4 bits of data can be put out by one (eg: D0 - D3) and read by the other (eg: S3 - S6); the other bit (eg: D4/S7) can be used for synchronizing (eg: Data Ready). If data is transferring one way, the Dn/Sn path the other direction can be used for acknowledgements. In this scheme, you must remember that S3 - S6 read the inverse of the other computer's D0 - D3 data bits, but S7 has the same value as D4 (or D3 - D6 and D7 respectively, for the alternate wiring). Also remember that S6 and S7's mapping to pins is swapped in order from the others.

16. Transfer Modes and Cables

Mode 1A: nibble mode, using Data Out to Status In connection This version works with all parallel ports; commercial xfer software style.

```
Side 1   Pin   dir   Pin   Side 2   connection
------   ---   ---   ---   ------   ----------

D0       2     =>    15    S3+      direct

D1       3     =>    13    S4+      direct

D2       4     =>    12    S5+      direct

D3       5     =>    10    S6+      direct

D4       6     =>    11    S7-      inverted
```

S7-	11	<=	6	D4	inverted
S6+	10	<=	5	D3	direct
S5+	12	<=	4	D2	direct
S4+	13	<=	3	D1	direct
S3+	15	<=	2	D0	direct
Gnd	25	===	25	Gnd	(ground)

Mode 1B: nibble mode, using Data Out to Status In connection This version works with all parallel ports; bit positions matched.

Side 1	Pin	dir	Pin	Side 2	connection
D3	5	=>	15	S3+	direct
D4	6	=>	13	S4+	direct
D5	7	=>	12	S5+	direct
D6	8	=>	10	S6+	direct
D7	9	=>	11	S7-	inverted
S7-	11	<=	9	D7	inverted
S6+	10	<=	8	D6	direct
S5+	12	<=	7	D5	direct
S4+	13	<=	6	D4	direct
S3+	15	<=	5	D3	direct
Gnd	25	===	25	Gnd	(ground)

Mode 1C: nibble mode, using Data Out to Status In connection; Controls wired for additional interfaces. This version works with all parallel ports.

Side 1	Pin	dir	Pin	Side 2	connection

D3	5	=>	15	S3+	direct
D4	6	=>	13	S4+	direct
D5	7	=>	12	S5+	direct
D6	8	=>	10	S6+	direct
D7	9	=>	11	S7-	inverted
S7-	11	<=	9	D7	inverted
S6+	10	<=	8	D6	direct
S5+	12	<=	7	D5	direct
S4+	13	<=	6	D4	direct
S3+	15	<=	5	D3	direct
C0-	1	<=>*	1	C0-	direct
C1-	14	<=>*	14	C1-	direct
C2+	16	<=>*	16	C2+	direct
C3-	17	<=>*	17	C3-	direct
Gnd	25	===	25	Gnd	(ground)

* Note: Control Out bits on receiver set high (including inversion, ie: C0,C1,C3=0; C2=1). Control feedback on receiver can read control out from sender. Can use some lines each way, and could switch C0 - C2 and C1 - C3 for symmetry if we want two lines each way, or other variations. Mode 2: 8 bits, using bidirectional parallel port This version works only with bidirectional parallel port whose Data Out can be tristated; the receiving side must tristate its Data Out port to use its feedback register as an 8 bit input port.

Side 1	Pin	dir	Pin	Side 2	connection
D0	2	<=>*	2	D0	direct
D1	3	<=>*	3	D1	direct
D2	4	<=>*	4	D2	direct
D3	5	<=>*	5	D3	direct
D4	6	<=>*	6	D4	direct

D5	7	<=>*	7	D5	direct
D6	8	<=>*	8	D6	direct
D7	9	<=>*	9	D7	direct
C0-	1	=>	13	S4+	inverted
C1-	14	=>	12	S5+	inverted
C2+	16	=>	10	S6+	direct
C3-	17	=>	11	S7-	direct
S4+	13	<=	1	C0-	inverted
S5+	12	<=	14	C1-	inverted
S6+	10	<=	16	C2+	direct
S7-	11	<=	17	C3-	direct
Gnd	25	===	25	Gnd	(ground)

* Note: bidirectional cards only; receiving side must tri-state with C5=1 If a two bidirectional ports are left connected in this fashion, and they are both enabled (eg: after powerup or reset) with different data outputs, then the 74LS374 driver chips could be "fighting". Just to be careful, when I created a cable like this (actually, a DB25 jumper box usually sold for RS-232 jumpering, along with straight through 25 line DB-25 cables), I used 8 10K resistors between the corresponding Data lines, to limit current in this case. (Actually, a DIP resistor pack fit perfectly on the PC board inside the DB-25 jumper box). The resistors are large enough to keep TTL output from overstressing another one if both enabled, but when one is disabled and the other enabled, the resistors are low enough to allow the TTL output to drive a TTL input well enough. Mode 3A: 8 bits, using Open Collector Control Outputs as inputs This version uses 4 control outputs as inputs, plus 4 status inputs.

Side 1	Pin	dir	Pin	Side 2	connection
------	---	---	---	------	----------
D0	2	=>*	1	C0-	inverted
D1	3	=>*	14	C1-	inverted
D2	4	=>*	16	C2+	direct
D3	5	=>*	17	C3-	inverted
D4	6	=>	13	S4+	direct
D5	7	=>	12	S5+	direct

D6	8	=>	10	S6+	direct
D7	9	=>	11	S7-	inverted
C0-	1	<=*	2	D0	inverted
C1-	14	<=*	3	D1	inverted
C2+	16	<=*	4	D2	direct
C3-	17	<=*	5	D3	inverted
S4+	13	<=	6	D4	direct
S5+	12	<=	7	D5	direct
S6+	10	<=	8	D6	direct
S7-	11	<=	9	D7	inverted
Gnd	25	===	25	Gnd	(ground)

* Note: Control outputs used as inputs must be programmed high: C0, C1, C3 = 0 and C2 = 1 Mode 3B: 8 bits, using Open Collector Control Outputs as inputs This version uses 3 control outputs as inputs, plus 5 status inputs; remaining control output is bidirectional - if left high by default, either side can pull low (remember inverted logic).

Side 1	Pin	dir	Pin	Side 2	connection
D0	2	=>*	1	C0-	inverted
D1	3	=>*	14	C1-	inverted
D2	4	=>*	16	C2+	direct
D3	5	=>*	15	S3+	direct
D4	6	=>	13	S4+	direct
D5	7	=>	12	S5+	direct
D6	8	=>	10	S6+	direct
D7	9	=>	11	S7-	inverted
C0-	1	<=*	2	D0	inverted
C1-	14	<=*	3	D1	inverted
C2+	16	<=*	4	D2	direct

```
S3+    15    <=*    5    D3    direct

S4+    13    <=     6    D4    direct

S5+    12    <=     7    D5    direct

S6+    10    <=     8    D6    direct

S7-    11    <=     9    D7    inverted

C3-    17    <=>   17    C3-   direct (OC shared)


Gnd    25    ===   25    Gnd   (ground)
```

* Note: Control outputs used as inputs must be programmed high: C0, C1, C3 = 0 and C2 = 1 [A future version of this document may sketch out the code to send and receive data through these connections]

17. Capturing "printed" data from another machine

A computer with a bidirectional printer port, connected with a Mode 2 cable to any standard port, could potentially pretend to be a printer so as to capture the "printed" information. It would configure its Data port for input, and set appropriate values on its Control Out to mimic a printer on the other computer's Status In lines. In the general case, the problem would be detecting the very brief -Strobe pulse; this would either require an external TTL latch triggered by -Strobe (either edge), or some way to sense a quick pulse on that line. In the latter case, a revised connection (call it Mode 2B) could connect the "printing" computer's -Strobe (C0-) line to the receiving computer's -Ack (S6+) line; the trailing edge of the printing computer's -Stobe would generate an interrupt on the receiving computer. I have not tried this. Of course, the +Busy line would also be needed to avoid data overflow; perhaps it could be kept high (busy) most of the time, but pulsed low after reading the data (which would be handled by the IRQ routine). [If anybody has had some success with this, let me know.]

18. Controlling Outputs

This can be easy; just use the Data Out TTL signals to control TTL level items. Unfortunately, they cannot source much current (providing positive voltage on the pin, relative to ground) - be careful of the 2.6 mA limit. Some LSI implementations might allow more or less than this (likely less).

```
        Dn Out ------+

                     |+

Sourcing            Load (up to 2.6 mA @ 2.4 v)

                     |-

        Ground ------+
```

If you have an external +5 volt supply, you have two options: use the Data Out pins to sink up to 24 mA from your +5 volt supply, or use buffer chips to control (source or sink) more current (or higher voltages). I have used an exteranl 5 Volt supply (regulated wall wart) plus optocoupled solid state relays as the "load", to control AC voltages (keep the high AC voltages away from any of this logic level stuff, obviously).

```
                         +------------------------------- (+5 v)

                         |+

   Sinking             Load (up to 24 mA @ 4.2v)

                         |-                               Power Supply

        Dn Out ------+


        Ground --------------------------------------- ( Gnd)
```

Use limiting resistors if you need to limit the current. If the load were an LED (or optocoupler) through which you wished to put 20 mA, do the calculations. The Dn Output will probably be around 0.7 volts, so you have about 4.3 volts of drop; the LED will drop about 1.9 v (check specs!), leaving 2.4 V to be dropped by a resister at 20 mA: 120 ohms. Test and measure, adjust to fit. You can also use the Control Out pins. They can't source much of anything (about 1 mA through the 4.7K resistors to +5), and can only sink about 7mA. (The LS TTL gate actually sinks 8 mA, but one is taken up by the 4.7 K resistor to +5). Again, check on clones with different electical specs. This can control TTL inputs fine, and might be able to run an optocoupler or solid state relay in sink mode (depends on the device). In one application, I used two 74ACT374 latches, which can source 48 mA or sink 64 mA. I connected the 8 inputs of each to the Data Out, and the latch clocks to two Cn outputs. In software, I put out 8 bits of data on the Data Out port, pulsed a Cn bit to latch it into one 74ACT374, put the next 8 bits out on Data Out, and used the other Cn bit to latch it into the other 74ACT374 - voila, 16 bits of 64mA output control. Of course, this took a separate +5 V power supply ($5 surplus regulated wall wart). If you can still find an old TTL parallel port (especially with sockets), you can substitute the 74ACT374 chip for the original 74LS374 and get better drive capability. The back of magazine suppliers were selling *fully socketed* TTL based parallel ports for about $15 a few years back; by cutting a trace and soldering a jumper you could make them bidirectional; by plugging in a $1 chip you could make them source/sink 48/64 mA. You might still find such TTL parallel port cards in old PCs. Typically, one designed for the original PC will still work fine on the ISA bus of your 486DX2-66, so don't worry about that.

19. Sensing switches

There are several ways to sense external switches via a parallel printer port. If you are dealing with naked switches, the simplest is actually to connect up to 4 switches between the control outputs (pins 1,14,16, and 17) and ground, program the control outputs high (counting inversions), and use the control feedback register to read switch state (counting inversions). This is because the pull-up resisters are already implemented for the open collector Control Out pins.

```
        Cn Out ------+           (Program output high; read feedback)

                     |

                   switch

                     |
    Ground ------+
```

Another simple method would be to use up to 5 pullup resistors (4.7K) from the Status inputs to +5 volts, and use switches to pull these down to ground. One disadvantage of this is that it uses an external +5 supply. TTL inputs tend somewhat to float high, so you *might* get by without the pullup resistors, but you are pushing it.

```
                 4.7K

    Sn In -------+-^^^^^--- (+5 v)        (Read status register)

                 |

               switch

                 |

    Ground ------+---------- (gnd)
```

You could use this same scheme for 8 inputs on the data lines of a bidirectional printer port - but you have to be sure that the outputs are tristated beforehand, or your switches may damage the 74LS374 (or equiv). This latter may be a problem after booting or rebooting. One approach is to use current limiting resistors in series "just in case". You should ideally still have pullup resistors too. (The 1K value is chosen for LS TTL, to keep the 2.4 V output at no more than 2.4 mA sourcing).

```
                  1K             4.7K

    Dn Out --------^^^^^-----+----^^^^^------ (+5v)

                             |

                           Switch

                             |

    Ground ------------------+-------------- (Gnd)
```

20. Optocoupled inputs

To be written.

21. Matrix scanning

In an upcoming project, I want to use a 4x4 keypad as an input. In this type of keypad (warning: there are other types), each key makes a connection from one row to one column, when pressed. My first design has the "columns" connected to the 4 Control Out pins, while the "rows" are connected to 4 of the Status Inputs. Normally, all 4 control outputs were programmed high, but to scan the keypad, I would lower one control output at a time, and read the 4 status inputs at that time. The 4 status inputs are each pulled high to +5 volts via a 10K resistor, or left to float. (Use Dn plus resistors rather than +5 external supply?) This scheme could be expanded to as large as 4x5 (4 Cn outputs, 5 Sn inputs).

```
                          10 K

S4 ------------X X X X--^^^^^--+--- (+5)

S5 ------------X X X X--^^^^^--+

S6 ------------X X X X--^^^^^--+     (these pullups optional?)

S7 ------------X X X X--^^^^^--+

               | | | |

C3 ------------+ | | |      Scan one Cn low at a time, read Sn each time

C2 --------------+ | |

C1 ----------------+ |

C0 ------------------+
```

Alternate scheme. Use Dn's for scanning output rows, through diodes which will isolate them from each other if two keys in the same column are pressed. (Otherwise we have drivers fighting, and indeterminate voltage levels). Germanium small signal diodes have less voltage drop. Use the Cn pins as inputs (with existing pullups) - program Control Out for high TTL levels, and read the feedback register for input (counting inversions). No extra power supply is needed. The diode between TTL output low and TTL input low will push the TTL noise margins, but you can usually get by with it. This scheme could be expanded up to 8x4 (8 Dn outputs, 4 Cn used as inputs). If we know that only one key (or one key per column) will be pressed at a time, the diodes can be omitted.

```
      1N914

D0 ----|<|-----X X X X      (Set all Cn high),

D1 ----|<|-----X X X X      Scan one Dn low at a time, read all Cn

D2 ----|<|-----X X X X

D3 ----|<|-----X X X X

               | | | |

C3 ------------+ | | |

C2 --------------+ | |

C1 ----------------+ |

C0 ------------------+
```

Note that if we press three switches in the matrix which form 3 corners of a box, it will appear as if all 4 corners are pressed; this is true of any such matrix, unless each switch has its own diode. I hope to have some results to share in a later revision of this doc.

22. Tutorial on TTL outputs

In response to requests, here is a brief tutorial on chip outputs. As a preface, it is conventional to discuss current as flowing from positive to negative, even though we know full well that electrons actually move in the opposite directions; it's just a convention, not ignorance.

```
     +5                                          +5

    /                                             \

---H                                              /   external resistor

    \                                             \

     |_____                            ___________/

     |   out                           |   out

    /                                 /

---L                              ---L

    \                                 \

    Gnd                               Gnd


Totem Pole                        Open Collector
```

Regular TTL outputs basically consist of a two "stacked" transistor in series between +5 volts and ground, with the output coming from the connection between them. This is called a "totem pole output". At any given time one of these transistors is conducting and the other is not. To pull the output "high", the transistor from +5 to the output conducts (H), which "sources" positive current from the output to ground (that is, an external device between the output and ground will get power). To pull the output low, only the lower transistor (L) conducts, "sinking" current to ground; an external device between +5 volts and the output can be energized. Current flows into the chip for low, out of it for high.

```
     +5

    /    |

---H on  V

    \      -->

     |________        TTL output on = 1 = high, "sourcing" current

     |   out  \

    /         / |

---L off      \ V

    \_________/

    Gnd
```

```
   +5_________
    /         \
---H off      / |
    \         \ v
     |________/      TTL output off = 0 = low, "sinking" current
     |  <-- out
    /
---L on |
    \   v
    Gnd
```

The 0 / pull low current capacity (sinking) is larger than the 1 / pull high capacity (sourcing). If you want to drive something like an LED, or a solid state relay, you can get more current from the TTL outputs by connecting it between +5 and the gate output (second picture) - *if* it's electrically isolated from ground. You still have to check the current and voltage ratings, tho. One key here is that the chip is always trying to pull either high or low, and the currently conducting transistor has voltage and current limits, beyond which it can be damaged. It is not good design to connect two such outputs which may have different states - one pulling high and one low - because this will exceed the current specs. However, if you do, the one pulling low will "win", since TTL can sink (pull low) more strongly than it can source (pull high). And there is some slack in the specs, so this does not always immediately damage the chip; we'll get back to this. These are the type of outputs on the DATA lines on the original IBM parallel port. (Well, not exactly, but I'll get back to that too). Another type of output is the "open collector". In this case, there is a transistor from the output pin to ground, but none to +5 volts. The two states are 0, "conducting to ground" (pulling low), and 1, "not conducting" (floating, not pulled either way). Externally, you can connect a resistor of proper value between the line and +5 volts to pull the line high when the chip output is floating. The value is chosen such that when the output is conducting to ground, it overpowers the external resistor and the line goes low. The advantage of this scheme is that you can connect multiple open collector outputs together (or even slip in one totem pole). Every output that is floating is ignored (and the resistor will pull the line high if and only if all outputs are floating); multiple outputs pulling low cause no conflict. This is the type of output used for CONTROL lines in the original IBM parallel port. Since each parallel port control output line has a corresponding feedback bit that can be read (mainly for diagnostics, to see if the line really goes high or low), it is possible to program these CONTROL outputs "high" (really floating), and then allow external signals control the high or low state of the line. An external open collector output, or a totem pole one, is capable of pulling the line high (reinforcing the pullup resistor) or low (overpowring it) without exceeding current specs. In this way you can use the control lines as inputs. If there is no external signal, it will read high. However, you must program the CONTROL outputs high (taking into account any logical inversions between the register bits and the electrical outputs); if the open collector output is low (conducting to ground), the external signal won't be able to pull it high (or at least not without exceeding the specs). The same thing can be done with totem pole outputs - program them high, and let external logic pull them high (no overloading the H transistor, rather than a pullup resistor, and exceeding spec, with possible unreliability or damage. However, this has been successfully done with the data outputs of a conventional parallel port, and some people claim not to have seen any damage yet. Don't blame me if you do it and something dies. A third type of gate is a totem pole in which both high and low transistors can be non-conducting at once, creating a third, floating state (this is often called a tri-state output). When it is in this third state, the line is not being pulled high nor low by this device, and thus can be safely controlled by some other device. No pullup resistor is used. The 74LS374 chip used for the standard parallel port DATA outputs actually has tristate ability, but as described elsewhere the third state is not used, and it is always trying to pull high or low - thus my initial description of the DATA lines as totem pole drivers. The way true bidirectional parallel ports work is to allow the software to selectively put the DATA outputs into the third, undriven state. Then you can use the data feedback register to read whatever highs or lows are put onto the data lines externally. Non-TTL logic, like CMOS, has the equivalents of these, but with a different type of transistor, and different voltage and current values. If your parallel port uses a single chip or otherwise differs from the original IBM parallel port, the above electrical description cannot be guaranteed, but it is probably pretty analogous. Let me know if you have specific knowledge about the electical specs of your single chip parallel port, for future versions of this document. Note again that high and low in this description are electrical levels, as would me measured by a voltmeter on the pins; high is associated with TTL logic 1, but whether a given CONTROL output line is high when the corresponding register bit is 1 or 0 varies with the line, as is described elsewhere in this document. The DATA lines are straight through, with no inversions, so a 1 bit produces a high output.

---

23. Contributors and Information Needed for future revisions

My thanks for information, proofreading and suggestions from: scharf@mirage.nsc.com (Steven M. Scharf) cgd@ecmwf.co.uk (Dick Dixon) rstevew@armory.com (Richard Steven Walz) geo@netcom.com (George Pontis) jmorriso@bogomips.ee.ubc.ca (John Paul Morrison) For the future: I would like specs on the various enhanced printer ports and chipsets. I would appreciate information on how parallel port connected Ethernet interfaces, SCSI interfaces, tape backup units, sound cards, modems, and copy protection devices work (which pins used how), including how "transparency" is achieved (or not) on those with "passthrough". I would appreciate info on data transfer programs, cables, and protocols. I would like to hear about interrupt driven PC printer drivers, for DOS or Unix or whatever (or why they don't work). I would like to hear of "printer capture" software and hardware that uses printer ports. I would like to know about differences between "old standard" TTL parallel ports and the LSI implementations (specific chip specs included). I would like to hear of your interfacing projects, sucessful or not. And of course, I'd like to hear about bugs and inaccuracies in this document. (Gently)

24. Differences

Version 0.92 corrected two minor typos which were kindly brought to my attention (D2 in place of C2 in one place, and 7505 in place of 7405), and reformatted some long lines. Version 0.94 added a tutorial on TTL outputs. Version 0.95 corrected the section on data xfer cables for Laplink style programs. Version 0.96 corrected still more typos. = end of Version 0.96 9/1/94 Zhahai Stewart parport@hisys.com